RL-TR-97-49
Final Technical Report
July 1997


# EMPIRICAL EVALUATION OF KBSA TECHNOLOGY

Andersen Consulting

William C. Sasso

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*



Rome Laboratory
Air Force Materiel Command
Rome, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-97-49 has been reviewed and is approved for publication.

APPROVED: Nancy A. Roberts
NANCY A. ROBERTS
Project Engineer

FOR THE COMMANDER: John A. Graniero
JOHN A. GRANIERO, Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL/C3CA, 525 Brooks Rd, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

| REPORT DOCUMENTATION PAGE | Form Approved OMB No. 0704-0188 |
|---|---|



| 1. AGENCY USE ONLY *(Leave blank)* | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | July 1997 | Final Sep 93 - Sep 96 |

**4. TITLE AND SUBTITLE**

EMPIRICAL EVALUATION OF KBSA TECHNOLOGY

**5. FUNDING NUMBERS**

C - F30602-93-C-0198
PE - 62702F
PR - 5581
TA - 27
WU- 70

**6. AUTHOR(S)**

William C. Sasso

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

Center for Strategic Technology Research
Andersen Consulting
3773 Willow Road
Northbrook, IL 60062-6212

**8. PERFORMING ORGANIZATION REPORT NUMBER**

N/A

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

Rome Laboratory/C3CA
525 Brooks Rd.
Rome, NY 13441-4505

**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

RL-TR-97-49

**11. SUPPLEMENTARY NOTES**

Rome Laboratory Project Engineer: Nancy A. Roberts/C3CA/315-330-3566

**12a. DISTRIBUTION AVAILABILITY STATEMENT**

Authorized for public release; distribution unlimited.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 200 words)*

This report evaluates the potential impact of Knowledge-Based Software Assistant (KBSA) Technology on software development productivity and software quality, and is submitted by Andersen Consulting to Rome Laboratory as a deliverable under contract F30602-93-C-0198. The empirical study - including both controlled experiments and a survey of industrial field experience - compared the impact of KBSA functionality to state-of-the market software development technologies. An experiment was conducted to evaluate the impact of core KBSA Technology: evolution transformations. As a benchmark, Andersen compared the productivity and software quality impacts of this technology with that of the level of functionality present in leading object-oriented Computer-Assisted Software Engineering (OO-CASE) tools. The data collected lead to a conclusion that KBSA Technology does have a positive and observable impact on software development productivity and software quality.

**14. SUBJECT TERMS**

Empirical Evaluation, Software Development

**15. NUMBER OF PAGES**

56

**16. PRICE CODE**

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UL |

## Executive Summary

This report evaluates the potential impact of Knowledge-Based Software Assistant (KBSA) technology on software development productivity and software quality, and is submitted by Andersen Consulting to Rome Laboratory as a deliverable under contract F30602-93-C-0198.

Andersen Consulting has conducted an empirical study of the impact of KBSA and related technologies on software development productivity and software quality. This study — including both controlled experiments and a survey of industrial field experience — compared the impact of KBSA functionality to state-of-the-market software development technologies.

In the second half of 1994, we conducted an experiment to evaluate the impact of a core KBSA technology: evolution transformations. As a benchmark, we compared the productivity and software quality impacts of this technology with that of the level of functionality present in leading object-oriented Computer-Assisted Software Engineering (OO-CASE) tools. The experiment was designed to investigate productivity, software quality, and time-to-completion of software projects. As a by-product of the experimental process, we collected data on the usability of KBSA technology.

The data lead us to conclude that KBSA technology does have positive and observable impacts on software development productivity (on the order of a factor up to 2.7). Similarly, KBSA technology improves software quality; improvements (on the order of a factor up to 1.15) were observed. However, this study has not shown that KBSA technology has a positive and observable impact on software time-to-completion.

In early 1994, we conducted an extensive electronic mail survey of the Knowledge-Based Software Engineering[1] (KBSE) community to identify practical applications of KBSE technology in industry and the Government. We were able to identify real-world applications of this technology in the following companies: Andersen Consulting, AT&T Bell Labs, Boeing Computer Services, Bull Honeywell, IBM (Canada), and Motorola. We collected further information on these applications, and organized a panel discussion of these application experiences at the September 1994 KBSE Conference in Monterey, CA. From these sources, we hear that KBSE technology is providing value in industry use today. Companies like Andersen Consulting , Boeing Computer Services, IBM/Canada, Motorola, and AT&T Bell Labs are taking advantage of the potential of this technology, and are presenting their results in public discussion.

[1]Within this report, KBSA technologies should be considered a subset of KBSE technologies.

The major findings of this study are these:

- KBSA technology demonstrates an experimentally observable positive impact on software development productivity (an improvement factor of up to 2.7). This is consistent with the observations of field use, especially the experiences reported by Andersen Consulting.

- KBSA technology demonstrates an experimentally observable positive impact on software quality (an improvement factor of up to 1.15). This is consistent with the observations of field use, especially the experiences reported at Motorola and Andersen Consulting.

- KBSA technology is usable. Our subjects, including software development professionals, military personnel, and graduate students, learned to use the powerful capabilities of the KBSA Concept Demonstration System effectively within a two-day training period.

These improvement factors may actually understate KBSA's potential impact significantly, as experimental design constraints made it impossible to include some KBSA capabilities in the experimental study. The most important lessons learned in this project are these:

1. Software experimentation is hard, but it can be done. As other studies have suggested, our experience emphasizes the need to control for different individual skills and learning rates in the experimental design.

2. Higher-order KBSA capabilities, such as the Bundle and Attribute-to-Relation evolution transformations, provide biggest productivity impact, but are perhaps least generic. This implies the need to study the specific requirements of users as a basis for designing these transformations, at least until we have established a set of generally applicable higher-order transformations.

3. This study has highlighted the importance of usability aspects in the design and implementation of KBSA technology. The suggestions made by the experimental subjects regarding usability indicate that KBSA technology has further to go in this dimension: KBSA is usable today, but it can be far more usable in the future.

## 1. Introduction

This report presents an evaluation of the potential impact of Knowledge-Based Software Assistant (KBSA) technology on software development productivity and software quality, and is submitted by Andersen Consulting to Rome Laboratory as a deliverable under contract F30602-93-C-0198.

### 1.1 General Motivation

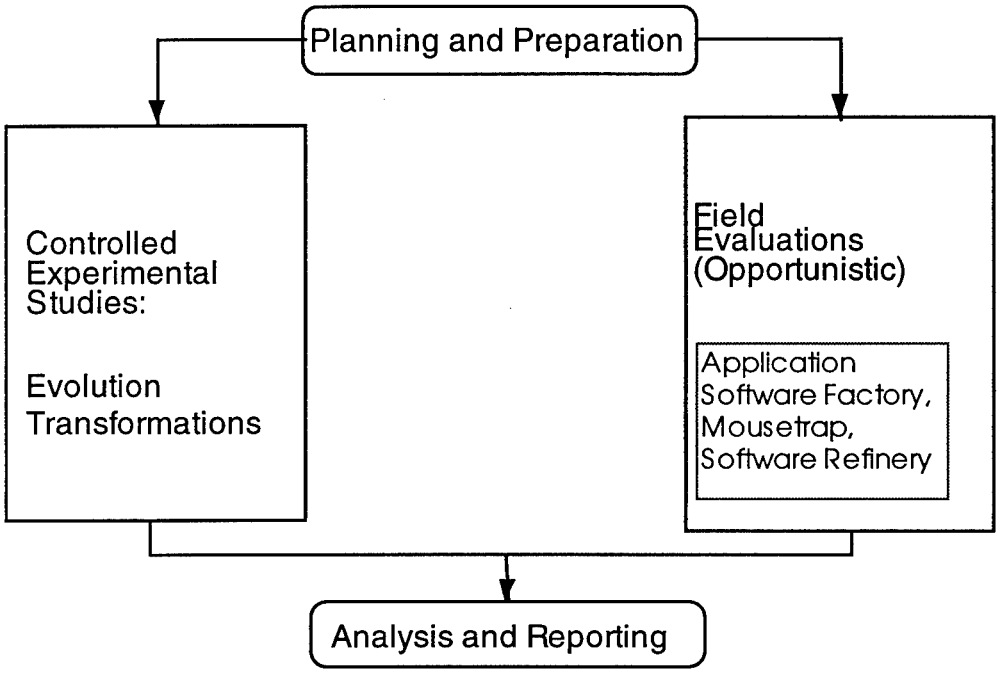
Throughout both the Department of Defense and the Software Engineering research community, an appreciation has grown of the need to balance the development of new Software Engineering (SE) techniques and technology with a greater ability to evaluate — in a systematic, scientific fashion — the relative merits of the current set of SE technology. For example, in the keynote speech at the 1992 International Conference on Software Engineering, Professor Nancy Leveson stated

> There are two stages in the early years of a new technology: (1) exploration of the space of possible approaches and solutions to problems (i.e., invention) and (2) evaluation of what has been learned by this trial and error process to formulate hypotheses that can be scientifically and empirically tested in order to build the scientific foundations of the technology. Most of our emphasis so far has been in the first stage (i.e., invention); it is time now to give more attention to the second. [Le92, p. 7]

In the same year, Mr. Lloyd Moseman (USAF) stated "I believe the time has come for us to cease our admiration of the metrics problem, and to start implementing some metrics." [Mo92a, p. 12] More recently, he has written that "Our collective challenge is to use measurement as a means to help software development move from an art to a science, or even an engineering discipline. It is difficult to improve software quality if you don't have the numbers to make a business case for investing in quality." [Mo92b, p. 5] Similarly the draft DoD Software Technology Strategy [DoD91] recognized the need for more empirical evaluation of software technology.

Like other DoD research programs, Rome Laboratory's Knowledge-Based Software Assistant (KBSA) effort faced this challenge. In response, Rome identified empirical assessment as one of the goals of the KBSA Advanced Development Model (ADM) project [RL92]. The results of ADM assessment, however, remain in the future. Fortunately, some KBSA technology (e.g. [De92a] and [Ko92]) is sufficiently mature and robust to support preliminary empirical evaluations. These partial studies can inform and guide the design of the ADM assessment, even as they provide empirical data within the short term.

As described in this report, Andersen Consulting has conducted an empirical study of the impact of KBSA and related technologies on software development productivity and software quality. This study included both controlled experiments and industrial field experience. It compared the impact of KBSA functionality to that of current software development technologies. Figure 1 depicts the high-level plan of the project.




### 1.2 Contribution of this study to KBSA Program Objectives

In 1983, Rome Laboratory's KBSA program proposed a new software development paradigm designed to provide dramatic increases in software development productivity and software quality [Gr83], [Wh91]. Technical progress has been made, and within the next year the KBSA Advanced Development Model (ADM) will be available for field trial. However, a major factor in motivating organizations to adopt new technologies is the observability of their benefits, as noted originally by [Ro83] and discussed in the specific KBSA context by [Sa91a] and [Sa91b]. This empirical evaluation of KBSA impact on software development productivity and software quality provides hard data to organizations considering the adoption of KBSA technology. In addition, this project has gathered information on experimental design and instrumentation techniques that can be used in support of the empirical evaluation component of the KBSA ADM project.

### 1.3 Objectives and Scope of the Study

As initially conceived, the study was intended to consist of two tracks: (1) a set of controlled experiments and (2) a set of opportunistic field observations. The experiments were designed to investigate the impact of core KBSA features on software quality and development productivity. These core features include, for example, evolution transformation technology and the automated capture and maintenance of traceability information. A recent KBSA implementation, the Concept Demonstration System, has brought these technologies to levels of robustness and usability enabling their systematic evaluation.

The second track, consisting of opportunistic field studies, was designed to complement the first by providing evidence and insight concerning the transferability of the experimental results into actual software development practice. These industrial experience studies were opportunistic in that they were conducted as the information became available, and required flexibility on the researcher's part concerning the types of data collected and the degree of experimental control present.

#### 1.3.1 Experimental Study Track

In the second half of 1994, we conducted an experiment to evaluate the impact of evolution transformations, a core KBSA feature. As a benchmark, we compared the productivity and software quality impacts of the evolution transformation capabilities with the level of corresponding functionality present in leading object-oriented Computer-Assisted Software Engineering (OO-CASE) tools, such as IntelliCorp's ProKappa [PK91a, PK91b]. The experiment was designed to investigate these hypotheses:

- H1: Subjects using core KBSA functionality will complete the experimental task in less time than subjects using OO-CASE functionality. (productivity)

- H2: Subjects using core KBSA functionality will complete the maintenance task with fewer errors than subjects using the OO-CASE functionality. (quality)

- H3: Subjects using core KBSA functionality will complete the entire set of experimental tasks in less time than subjects using the OO-CASE functionality. (time-to-completion)

We compared the productivity of subjects using KBSA core functionality with that of subjects using capabilities equivalent to those present in today's leading OO-CASE tools. To control for potential confounding factors such as different development process models, graphical representations, menu

structures, and system terminology, we developed two versions of the KBSA Concept Demo: one with KBSA-level functionality enabled, and one with a lesser set of functionality enabled, representing the state-of-the-market OO-CASE commercial technology. Further, to control against the possible impact of previous development experience, we included not only software development professionals but also students with relevant backgrounds as subjects in the study. Further, we structured the data collection process to use natural controls against variances in factors such as experience and previous exposure to KBSA technology. A detailed description of the experimental design, and a discussion of its underlying rationale, will be presented in section 3 below.

### 1.3.2 Opportunistic Field Study Track

Experience in the trial application of Knowledge-Based Software Engineering (KBSE) technologies[1] — such as use of a wide-spectrum language (Refine), transformation capabilities, multiple views, and formal models of the software artifact — has great relevance both to this project and to the KBSA ADM assessment process. This approach can provide initial real-world evidence on the scale-up and value of KBSA technologies. Therefore it complements the experimental study described above. Further, these observations have reinforced our confidence in the results observed in the experiments.

In early 1994, we conducted an extensive electronic mail survey of the KBSE community to identify practical applications of KBSE technology in industry and the Government. In this manner, we were able to identify real-world applications of this technology in the following companies: Andersen Consulting, AT&T Bell Labs, Boeing Computer Services, Bull Honeywell, IBM, and Motorola. We collected further information on these applications via telephone interviews and analysis of published reports, and organized a panel discussion of these applications at the September 1994 KBSE Conference in Monterey, CA. An elaborated discussion of these applications and their impact is presented in section 2 below.

[1] For the purposes of this report, the term "Knowledge-Based Software Assistant" and the acronym KBSA will be restricted to use in reference to work sponsored by Rome Laboratory's KBSA research program, and the term "Knowledge-Based Software Engineering" and acronym KBSE will be used in general reference to all uses of knowledge-based technology to enhance software engineering technology capabilities. Thus, within this report, KBSA technologies should be considered a subset of KBSE technologies.

### 1.4. Proposal and Award

In response to PRDA Announcement 92-08-PKRD issued by Rome Laboratory, Andersen Consulting developed and submitted a proposal to conduct an empirical evaluation of KBSA technology. After evaluation by Rome Laboratory Andersen Consulting was awarded Government Contract F30602-93-C-0198 (dated September 21, 1993) to conduct the evaluation study reported here.

### 1.5 Acknowledgments

I would like to note my gratitude to the following people, whose assistance made the conduct and completion of this project possible:

• Mr. Ken Hu, a software engineering intern in Andersen Consulting's Center for Strategic Technology Research (CSTaR), who adapted the KBSA Concept Demonstration System to support the experimental data collection process;

• Professor Walt Scacchi of the University of Southern California, who provided the definition of the baseline OO-CASE functionality used as an experimental baseline;

• the original developers of the KBSA Concept Demonstration System, including especially Michael DeBellis, Kanth Miriyala, Sudin Bhat, Guillermina Cabral, and Steve Wagner, all (current or former) members of the Software Engineering Laboratory at CSTaR;

• Professor Paul Bailor of the Air Force Institute of Technology, Professor Bala Ramesh of the Naval Postgraduate School, and Professor Perry Alexander of the University of Cincinnati, and their students, who participated in this study as subjects;

• Kevin Benner, Ph.D., and Gerry Williams, Ph.D., colleagues at Andersen Consulting who provided comments and suggestions regarding the organization of this study and the presentation of its results, and the Andersen Consulting personnel who participated in this study as subjects; and

• Mr. Douglas White and Ms. Nancy Roberts of the Knowledge-Based Software Engineering Group at Rome Laboratory, USAF.

## 2. Field Study Track

Experience in the trial application of KBSE technologies — such as use of a wide-spectrum language (Refine), transformation techniques, multiple views, and formal models of the software artifact — has great relevance both to this project and to the KBSA ADM assessment process. This approach can provide initial real-world evidence on the scale-up and value of KBSA technologies. Therefore it complements the experimental study described above. Further, these observations have reinforced our confidence in the results observed in the experiments.

### 2.1. Search for Applications of KBSE Technology

Under the auspices of this project, we conducted an extensive electronic mail survey of the KBSE community to identify practical applications of KBSE technology in industry and the Government. In this manner, we were able to identify real-world applications of this technology in the following companies: Andersen Consulting, AT&T Bell Labs, Boeing Computer Services, Bull Honeywell, IBM, and Motorola. We collected further information on these applications, and organized a panel discussion of these applications at the September 1994 KBSE Conference in Monterey, CA.

### 2.2. Sites Identified and Results Reported

In this section we will summarize the panelists' reports of how KBSE technology has been used in actual software development projects. In particular, they were asked to describe its impact on software development, by repsonding to the following questions.

- What are the "knowledge-based" aspects of the technology you have used?
- What kind(s) of application have you used KBSE technology to support?
- What have been the most positive impacts of using the technology? Reports of quantitative data were strongly encouraged.
- What obstacles or difficulties have you encountered in applying the technology? What do you feel can be done to remedy them?

The synopses presented below reflect actual use of KBSE technology in support of actual software development.

Bull HN Information Systems is currently investigating the integration of process-centered environments (e.g., Marvel) with collaboration environments (e.g., ConversationBuilder). After process model components in an earlier version proved difficult to modify, Marvel was chosen to

provide an integration layer within the application, and its KBSE capabilities are being used to enhance the flexibility and customizability of the application's process model [Ar94].

IBM/Toronto[2] has used Reasoning Systems' Software Refinery® to build an automated code inspection system for the PL/1 dialect used in SQL/DS (now known as DB2/VM), a major IBM mainframe database management system product. Since this particular dialect combines PL/1 and System 370 Assembly language, it requires the powerful language modeling capabilities of Refinery [BH91], [Tr92], [BH92], [HTB93].

As reported at the panel, Boeing Computer Services has used Reasoning Systems' Software Refinery® to develop a COBOL re-engineering facility. These tools have then been used to re-engineer large applications (e.g., 750 KLOC), and have been very successful at supporting the evolution of these applications. Members of the Boeing technical staff have stated that Software Refinery has a qualitative impact, enabling re-engineering work that would simply not be feasible without its support.

Andersen Consulting's Denver consulting office demonstrated the Knowledge-Based Design Assistant (KBDA) at the Andersen Consulting open house at KBSE-93 in Chicago. KBDA is an ART-IM application that automates the selection of code shells to improve the quality of the application while reducing the effort required to develop it. KBDA has been used to develop several client applications (each approximately 1,000 KLOC or larger), and has dramatically improved programmer productivity. Reported improvements are on the order of 5 to 1 [BSL93].

AT&T Bell Laboratories has worked with the Articulator process modeling software, applying it to represent the software engineering processes used by a large software development organization. When compared to the current process description facility, an on line library containing descriptions in natural language, the process descriptions produced using the Articulator approach were more succinct, more accurate, more complete, more easily understood, and more easily measured [Vo93]. This group also has knowledge of developers' experiences with AT&T Bell Labs' Designer Assistant [TSL].

Motorola has used transformation technology to produce code for real-time communications devices. Working from specifications (e.g., abstract data types), the technology has produced programs on the order of 2 KLOC to 3 KLOC . When compared with the code hand-crafted by engineers, the generated code has been found to (1) contain fewer errors, (2) be more compact, and (3) offer comparable (usually superior) performance. Non-

[2] Please note that this paragraph is derived from the cited works; unfortunately, no IBM/Toronto personnel were able to participate in the panel at the KBSE Conference.

published but publicly reported results include reductions in coding time of 3.6 to 1, elimination of code defects, and — perhaps most unexpectedly — reductions in code size by a factor of 1.2 to 1 [We94]. This latter is an especially important impact relative to embedded systems.

In summary, KBSE technologies have begun to move out of the research laboratory and into the real world of software development. While their use remains restricted in scope, the panelists' comments and their public reports certainly offer encouraging evidence that KBSE technology can both increase development productivity and improve software quality.

## 3. Experimental Study Track

As part of this project, we conducted an experiment to evaluate the impact of core KBSA features: (1) evolution transformations, and (2) automated maintenance of traceability information. The experiment was designed to investigate these hypotheses:

- H1: Subjects using core KBSA functionality will complete individual experimental tasks in less time than subjects using OO-CASE functionality. (task-level productivity)

- H2: Subjects using core KBSA functionality will complete the set of experimental tasks with fewer errors than subjects using the OO-CASE functionality. (quality)

- H3: Subjects using core KBSA functionality will complete the entire set of experimental tasks in less time than subjects using the OO-CASE functionality. (time-to-completion)

Thus the set of hypotheses includes both task-level impact and aggregated impact over the entire set of tasks in the experiment.

### 3.1. Comments on the Experimental Design

In planning the experimental design process, we identified several issues that our design needed to address:

- What do we compare Concept Demo against?
- How do we control individual subject variance?
- How do we control for the learning curve?

First, we considered the question "What do we compare Concept Demo against?" After evaluating the pros and cons of the possibilities, we chose to compare the Concept Demo against a "baseline" version of itself. In other words, we evaluated the productivity of subjects using KBSA core functionality with that of subjects using a lesser set of capabilities (referred to as the "baseline" capabilities). This enabled us to control for potential confounding factors between different software tools such as different development process models, graphical representations, menu structures, and system terminology. Therefore we developed two versions of the KBSA Concept Demo: one with KBSA-level functionality enabled, and one with the baseline set of functionality.

To determine what functional capabilities should be included in the baseline level of technology support, we engaged the services of Professor Walt Scacchi of the University of Southern California. Professor Scacchi, as a leading

authority on software engineering, is familiar with both the KBSA research program and the typical degree of support provided in "state-of-the-market" object-oriented Computer-Assisted Software Engineering (OO-CASE) technology, such as IntelliCorp's ProKappa [PK91a, PK91b]. As such, he was ideally situated to conduct an unbiased comparison of the relative capabilities of these technologies. We used his conclusions to allocate functionality between the baseline and full-KBSA versions of the Concept Demo.

We then turned to consideration of an issue that has been raised in earlier empirical software engineering research: the high degree of individual variation observed between subjects. This appears in many published reports of empirical studies, whether the subjects are software development professionals or students. Previous researchers have observed large differentials in individual productivity. These differentials — sometimes more than a factor of 2 — could confound the analysis of productivity data. As suggested by Lee [Le89], we have employed natural controls in our experimental design to guard against this situation. In particular, we have structured our design such that comparisons are made between data generated by a given subject rather than by different subjects. Further, to control against the possible impact of previous development experience, we included not only software development professionals but also students with relevant backgrounds as subjects in the study.

Finally, we concerned ourselves with the potential impact of the "learning curve" effect, another issue in this type of research. That is, the subjects are learning about the technology at a general level as well as the specific full-KBSA technology. For instance, in each hands-on exercise they become more aware of how to use the Concept Demo interface as well as how to apply evolution transformations. Thus, a possible explanation of improved subject productivity is that it is caused by the subjects' increasing familiarity with the tool in general rather than with the specific impact of any functional capability. To control for this, we designed the data collection procedure to counterbalance the availability of full-KBSA technology. Some comparisons were based on observations where the full-KBSA technology was used after the baseline technology, while others were based on the opposite sequence of observations. Thus, over the full set of data, the impact of any systematic learning curve effect could be expected to net to zero.

### 3.2. Design of Experimental Studies

The general design of the research included a standard sequence of training and data collection sessions. The training sessions, as the term is used here, included lectures, demonstrations, hands-on trial use sessions, and group discussion (e.g., questions and answers). In the schematic below, training sessions are indicated by $T_x$.

In each data collection session, subjects were given hands-on access to a specific set of KBSA technology (either the baseline or the full-KBSA versions of the KBSA Concept Demo). Data collection sessions, as we use the term, refers to a hands-on session in which a detailed sessions log is automatically generated by the Concept Demo, recording and timestamping the user's interface-level actions, as he or she performs a pre-specified set of software development tasks. In the schematic below, data collection sessions are indicated by $DC_x$.

The overall research design then can be represented as follows.

$$T_1 \rightarrow DC_1 \rightarrow T_2 \rightarrow DC_2 \rightarrow DC_3 \rightarrow T_3 \rightarrow DC_4$$

The Overall Research Design
(T = Training Session; DC = Data Collection Session)

In this representation, $DC_1$ and $DC_4$ represent "baseline" data collection sessions, while $DC_2$ and $DC_3$ represent "Full-KBSA" data collection. $DC_1$ and $DC_2$ are paired data collection sessions, meaning that an equivalent experimental task was performed, once with the support of baseline functionality (in $DC_1$) and again with Full-KBSA functionality (in $DC_2$). $DC_3$ and $DC_4$ are similarly paired, save that $DC_3$ (the Full-KBSA data collection session) in this case precedes $DC_4$ (the baseline data collection session). Appendix A includes a copy of the more detailed agenda for the two-day program.

### 3.3. Adaptation of the KBSA Concept Demo

We adapted the delivery version of the KBSA Concept Demonstration System to support this research design. In particular, we made the following modifications:

- We added a password-controlled switch enabling us to identify whether the subject had access to the baseline functionality or the full-KBSA functionality.

- We added a fine-grained session log feature, enabling the Concept Demo to capture the data describing the user's session.

- We developed task-specific knowledge-bases, containing the necessary hypertext requirements, and definitions of formal objects (e.g., classes, attributes, relations, etc.).

- We made robustification enhancements (e.g., by developing guards on the execution of evolution transformations).

- We reviewed the usability and consistency of the user interface, and made several interface-level changes to improve usability.

### 3.4 Design and Development of the Experimental Task-Sets

The design of the experimental tasks was constrained in several senses. For example, the set of capabilities present in the Concept Demo had to support the completion of the task-set, and had to do so in two different ways. First, with its feature switches set to the baseline level, the Concept Demo had to enable completion of the task-set in a manner corresponding to its completion in an OO-CASE tool. Second, with its feature switches set to "full-KBSA," the Concept Demo had to provide additional support for the task-set. In other words, the Concept Demo had to support two alternate task completion paths: one representing the capabilities of OO-CASE tools, and the other reflecting the incremental capabilities of KBSA. So, for example, we disabled some more sophisticated evolution transformations (e.g., Bundle, Define-Inverse, and Redefine-Cardinality) in the baseline version, while continuing to provide a set of lower level transformations sufficient to enable the subject to perform the experimental task.

There were other constraints as well. Very few of the subjects were familiar with formal specification languages, and the two-day training and data collection sessions did not allow for them to learn Extended Refine Specification Language (ERSLa). This restricted the feasible set of experimental tasks to those which could be performed using the Concept Demo's direct manipulation interface. This encouraged us to select some of the more robust Concept Demo interface presentations, such as the Hypertext views, the Entity/Relation diagram and the Class Hierarchy Diagram, as the basis for the set of experimental tasks.

There were several other constraints on the design of the experimental tasks. They had to be doable by motivated, trained users; we were not primarily interested in collecting data on the difficulty that users experienced with a piece of software as complex as the Concept Demo. This meant that each set of tasks had to be manageable. On the other hand, the larger the number of specification objects and the more complex the work to be done, the more realistic the experimental task would be. The tension between these two values led us to avoid the trivial in the design of the tasks, typically giving the subjects sets of twenty (20) tasks to be performed on pre-defined knowledge-bases containing up to fifty (50) formal specification objects.

After the experimental tasks were designed, the training materials and exercises were developed. They were then informally piloted and reviewed to ensure that the knowledge of the Concept Demo required to complete the tasks was included in the training. The training materials included primarily PowerPoint slides and handouts, and were supplemented by on-line

demonstrations of the Concept Demo software. A preliminary version of these materials was completed by mid-July, 1994.

### 3.3 Conduct of the Experimental Studies

In parallel with the design of the experimental tasks and the development of the training materials, we conducted a search for experimental subjects. We wanted to include both Department of Defense software development personnel and commercial software developers. The latter we recruited as volunteers from within Andersen Consulting, and the former we were fortunate to find through the assistance of faculty members at the Air Force Institute of Technology (AFIT) and the Naval Postgraduate School (NPS).

Between July and December, 1994, we conducted a series of five experimental studies. In each case, subjects participated in a formal training program (approximately eight hours of presentation and discussion) and performed a series of hands-on exercises, totaling approximately eight hours of time using the Concept Demo itself. A copy of the training program agenda is included as Appendix A.

#### 3.3.1. AFIT Pilot Study

With the cooperation and assistance of Professor Paul Bailor of the Air Force Institute of Technology (AFIT), we conducted a pilot experimental session at that location on July 19th and 20th, 1994. In the true spirit of pilot tests, the AFIT session enabled us to identify several critical issues in our training program and the experimental version of the Concept Demo. First, we revised the training to include a separate topic introducing the trainees to the Concept Demo interface, with an accompanying hands-on exercise to convey a sense of the interface operations and the Concept Demo's typical response time. We also made adjustments to the session logging mechanism, enabling us to track the subjects' interface-level activity at a finer level of granularity. We further revised several of the experimental task descriptions, and adapted the knowledge-bases that support those exercises.

The AFIT subjects were all Air Force officers in a graduate program in Computer Science. Most had some experience in software development. A profile of these subjects, including their previous level of exposure to KBSA technology, is presented in Table 3.4.2.1-1 (below). Due to the pilot nature of this study, no data was considered usable.

#### 3.3.2. AC1 Pilot Study

A second pilot session was conducted at Andersen Consulting's Center for Strategic Technology Research (CSTaR) in Chicago, IL, on August 24th and 25th. The AC1 subjects were all experienced Andersen Consulting employees

(software development consultants). Further adjustments to the experimental design were made as a result of this pilot study. A profile of these subjects, including their previous level of exposure to KBSA technology, is presented in Table 3.4.2.1-1 (below). Due to the pilot nature of this study, no quality data was considered usable.

### 3.3.3. NPS Experimental Study

With the cooperation and assistance of Professor Bala Ramesh of the Naval Postgraduate School (NPS) in Monterey, CA, we conducted a training and data collection session there on October 20th and 21st. This was a very successful study; we captured 45 matched-pair data sets of KBSA development productivity impact. These subjects were all military — mainly naval — officers in a graduate program in management information technology. A profile of these subjects, including their previous level of exposure to KBSA technology, is presented in Table 3.4.2.1-1 (below).

### 3.3.4. AC2 Experimental Study

Another training and data collection session was conducted at Andersen Consulting's Center for Strategic Technology Research (CSTaR) in Chicago, IL, on October 24th and 25th. The AC2 subjects were all experienced Andersen Consulting employees (software development consultants). A profile of these subjects, including their previous level of exposure to KBSA technology, is presented in Table 3.4.2.1-1 (below). This was a very successful study; we captured 22 matched-pair data sets of KBSA development productivity impact.

### 3.3.5. UC Experimental Study

With the cooperation and assistance of Professor Perry Alexander of the University of Cincinnati (UC) in Cincinnati, OH, we conducted a training and data collection session there on December 21st and 22nd. The UC subjects included one professional software developer and four graduate students in Electrical and Computer Engineering. A profile of these subjects, including their previous level of exposure to KBSA technology, is presented in Table 3.4.2.1-1 (below). This was a very successful study; we captured 27 matched-pair data sets of KBSA development productivity impact.

## 3.4. Collection and Analysis of Data

After discussing the techniques used to capture the data, we will present the productivity, quality, and time to completion data in summary form, describe the statistical techniques employed, and present our evaluations of the hypotheses in light of the data.

### 3.4.1 Data Collection Techniques

We used two major data collection mechanisms: (1) surveys completed by the subjects and (2) the Concept Demo's automated session log facility.

Survey forms were used to capture information about the subjects' relevant background knowledge and experience, usability data, and suggestions concerning alternative interface and functional capabilities that could be used in later KBSA technology.

The session log facility was added to the KBSA Concept Demo to support automated collection of developer productivity and software quality data. Subjects enabled this facility themselves (by invoking a menu option), so they were always aware when it was running. When they disabled the option, it wrote the current state of the knowledge-bases to the session log file. This gave us a time-stamped record of the actions invoked during the session, and the complete outcome of their work, in terms of the underlying formal specification of the knowledge-base contents.

### 3.4.2. Presentation of the Data

This section presents summarized (and in one case, normalized) data collected during the experimental sessions conducted at the Naval Postgraduate School (NPS), Andersen Consulting (AC2), and University of Cincinnati (UC).

#### 3.4.2.1. Background and Usability Data

After each hands-on exercise, the subjects completed short survey forms in which they rated the degree of ease or difficulty that they encountered in completing the exercise. This was a simple six-point scale from very easy to very hard, in response to the question "Overall, how easy or difficult was it for you to complete this hands-on exercise?" The data are summarized in the following table.

These data support the position that the subjects — in their own opinions — were in general able to use the KBSA Concept Demo software without material difficulty. Out of 102 reports, only one exercise was reported as "somewhat hard", and only sixteen were reported as "slightly hard." These reports were from subjects with a maximum of two days of hands-on experience with the software. While the experimenters' observations and comments from the subjects (as reported in Appendix B) make it clear that they identified aspects of the Concept Demo interface and functionality that could be improved, these data support the position that a suitable graphical interface will enable software development professionals to use KBSA technology in a comfortable and effective manner.

Averages by Group

| Familiarity with | AFIT | AC1 | NPS | AC2 | UC |
|---|---|---|---|---|---|
| 1. OO Development Concepts | 1.91 | 1.33 | 1.10 | 1.00 | 1.67 |
| 2. OO Development Experience | 1.14 | 0.56 | 0.80 | 0.67 | 1.17 |
| 3. CASE Experience | 0.82 | 1.11 | 1.00 | 1.00 | 0.67 |
| 4. Software Refinery® Experience | 0.95 | 0.00 | 0.00 | 0.00 | 1.17 |
| 5. KBSA Experience | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 6. KBSA Concepts | 0.91 | 0.00 | 0.40 | 0.00 | 1.50 |
| 7. Formal Methods Concepts | 1.23 | 1.22 | 1.10 | 0.33 | 2.00 |
| 8. Formal Methods Exprerience | 0.91 | 1.00 | 0.80 | 0.33 | 2.00 |
| 9. Years of SE Experience | 2.59 | 3.28 | 0.60 | 4.08 | 1.67 |

Key for questions 1 through 8: None = 0; Some = 1; Strong = 2; Very Strong = 3

TABLE 3.4.2.1-1
AVERAGE SUBJECT FAMILIARITY WITH KBSA

| | Very Easy | Somew't Easy | Slightly Easy | Slightly Hard | Somew't Hard | Very Hard |
|---|---|---|---|---|---|---|
| NPS | 15 | 16 | 5 | 11 | | |
| AC2 | 8 | 16 | 8 | 5 | 1 | |
| UC | 5 | 11 | 1 | | | |
| Total | 28 | 43 | 14 | 16 | 1 | |

TABLE 3.4.2.1-2:
USABILITY FEEDBACK

Further, these data suggest that the productivity and quality data presented below is valid in the sense that subjects expressed confidence that they had been able to perform the hands-on tasks for the data collection sessions in an adequate manner.

3.4.2.2. Productivity Data

For several of the hands-on exercises, subjects were instructed to enable the session log function in the experimental version of the Concept Demo software. This created a timestamped log of all user operations at the interface level and Concept Demo internal operations at the level of the knowledge base, enabling us to determine the time required to perform a

given task using one of the two alternative sets of KBSA capabilities (full-KBSA and baseline). In groups of two matched exercises, each subject used first one set of capabilities (either baseline or full-KBSA) and then the other to perform equivalent experimental tasks. The session logs enable us to compare the time required to perform these tasks with these different capabilities.

The data presented here are the (signed) differences between the time required to perform the task with full-KBSA capabilities and the time required with baseline capabilities. Thus negative values represent cases where the full-KBSA capabilities reduced the time required to perform a given task. Blank cells indicate that the subject's data was not usable for that task. This could mean, for example, that the subject used the same baseline functionality — which was always available — to perform the task, even though the advanced functionality was enabled.

These data have been normalized by expressing this difference as a percentage of the time required using full-KBSA capabilities, in order to filter the effect of differences in hardware and network capacity across the pool of subjects. Each subject worked on the same machine throughout the data collection process.

| NPS-1 | usr31 | usr33 | usr36 | usr37 | usr39 | usr40 |
|---|---|---|---|---|---|---|
| Bundle | -39 | -336 | -320 | -42 | -128 | -137 |
| Att->rel'n | | -207 | -396 | -573 | -66 | -105 |
| Create inv | | -141 | -87 | -222 | | -61 |
| Create inv | -115 | -162 | -134 | -306 | -62 | -234 |
| Att->rel'n | | -34 | -228 | -146 | -88 | -85 |
| Create inv | | -86 | -92 | -88 | -194 | |

TABLE 3.4.2.2-1:
NPS-1 PRODUCTIVITY DATA

Table NPS-1 above can be interpreted as follows. Each column displays the data representing one subject's experiences. For subject usr31, for example, we have only two matched pairs of data. From the value representing the difference between the first pair of matched values (that is, the value "-39"), we note that the full-KBSA technology allowed the subject to perform one of the experimental tasks more quickly: further, we can say that the task was completed 39% more quickly. Similarly, further down in the column, we note that Subject usr31 was able to perform a create-inverse operation 115% more quickly using the full-KBSA technology.

A few positive values can be observed; for example, several occur in the bottom row of Table AC2-2. These positive values indicate matched pairs of

data values where the subject performed the operation in less time using the baseline functionality. Over the full set of 99 matched pairs of data values, in 3 cases the baseline functionality enabled the subject to complete the operation in less time.

| NPS-2 | usr32 | usr37 | usr39 | usr40 |
|---|---|---|---|---|
| Att->rel'n | -75 | -89 | -83 | -135 |
| Att->rel'n | -133 | -93 | -263 | -274 |
| Bundle | | | -126 | -102 |
| Att->rel'n | -84 | -273 | -286 | -498 |
| Bundle | | -277 | | -346 |
| Create inv | -188 | -2 | -33 | -34 |

TABLE 3.4.2.2-2:
PRODUCTIVITY DATA: NPS-2

| AC2-2 | usr42 | usr43 | usr44 | usr45 |
|---|---|---|---|---|
| Att->rel'n | -123 | -134 | -71 | -52 |
| Att->rel'n | -88 | -170 | -72 | -130 |
| Bundle | -9 | -78 | | -16 |
| Att->rel'n | -211 | -72 | -65 | -58 |
| Bundle | -317 | -382 | | -210 |
| Create inv | -67 | +59 | +15 | -19 |

TABLE 3.4.2.2-3:
PRODUCTIVITY DATA: AC2-2

| UC-1 | usr 51 | usr 52 | usr 53 |
|---|---|---|---|
| Bundle | | -567 | -283 |
| Att->relation | -238 | -518 | -10 |
| Create inv | +49 | -22 | -86 |
| Create inv | -18 | -21 | -57 |
| Att->relation | | -553 | -82 |
| Create inv | -164 | -70 | |
| Create inv | -1 | -37 | |

TABLE 3.4.2.2-4:
PRODUCTIVITY DATA: UC-1

| UC-2 | usr51 | usr52 | usr53 |
|---|---|---|---|
| Att->relation | | -30 | -30 |
| Att->relation | | -357 | -68 |
| Bundle | -372 | -312 | -60 |
| Att->relation | | | -28 |
| Bundle | -531 | | -74 |
| Create inv | | | |

TABLE 3.4.2.2-5:
PRODUCTIVITY DATA: UC-2

Throughout these tables, the high predominance of negative values indicates that full-KBSA capabilities did in fact reduce the time required to perform the tasks. The high variances present in a given row can be attributed, at least in part, to individual variation between different subjects. The variance within a column may be explained, in part, by the differential power of the full-KBSA transformations: for example, the Bundle transformation performs more than does the create-inverse transformation, so the difference values for Bundle tasks were consistently higher than those for create-inverse tasks.

3.4.2.3. Data on Software Quality and on Time to Completion

As the final step in completing each exercise, subjects disabled the session log function , which invoked the copying of the current contents of the knowledge-base to the log file. In the analysis presented in the previous section, we compared the time required to perform specific, well-defined activities in the development of a formal specification. The comparison of the two versions of that specification — one developed with the baseline functionality and the other with the full-KBSA functionality — enables comparison of the degree of completeness and internal consistency of the matched pair of underlying specifications developed by each subject

| AC2 | Baseline Completeness | Time | Full-KBSA Completeness | Time | Difference Completeness | Time |
|---|---|---|---|---|---|---|
| usr42 | 88% | 1123 | 100% | 755 | 12% | -368 |
| usr43 | 69% | 3370 | 75% | 2558 | 6% | -812 |
| usr44 | 85% | 1478 | 100% | 1261 | 15% | -217 |
| usr45 | 88% | 1201 | 100% | 1133 | 12% | -69 |
| Averages | 83% | 1793 | 94% | 1427 | 11% | -366 |

TABLE 3.4.2.3-1:
AC2 QUALITY AND TIME TO COMPLETION DATA

| NPS | Baseline Completeness | Time | Full-KBSA Completeness | Time | Difference Completeness | Time |
|---|---|---|---|---|---|---|
| usr32 | 85% | 2015 | 100% | 2016 | 15% | +1 |
| usr37 | 85% | 1461 | 100% | 1934 | 15% | +473 |
| usr39 | 85% | 2103 | 92% | 1454 | 7% | -649 |
| usr40 | 85% | 1859 | 100% | 1323 | 15% | -536 |
| Averages | 85% | 1859 | 98% | 1682 | 13% | -178 |

TABLE 3.4.2.3-2:
NPS QUALITY AND TIME TO COMPLETION DATA

These tables should be read as follows. The second and third columns from the left ("Baseline Completeness" and "Time") contain data reporting the completeness (quality) and time to completion of subjects using the baseline capabilities to perform the complete experimental exercise. The fourth and fifth columns from the left ("Full-KBSA Completeness" and "Time")report the completeness and time to completion for that same subject, as he or she used the full-KBSA capabilities. The sixth column is the difference between the second and fourth columns, with a positive value indicating a higher degree of completeness was achieved with full-KBSA support. The seventh column is the difference between the third and fifth columns, with negative values indicating that the exercise was completed in less time using the full-KBSA capabilities.

Data in these tables is not available for all subjects. In some cases, subjects did not invoke the log-session capability, or did not do so correctly. In other cases, a subject may not have been present for both sessions required to perform the comparison. At the Naval Postgraduate School, for example, some subjects had other classes or school commitments to attend, forcing them to miss some of the exercises. That is, a subject may not have completed one of the hands-on exercises, rendering his or her log incomplete with respect to the quality and time-to-completion analyses. Finally, in some cases a subject encountered a termination condition or invoked the Concept Demo's "Abort" function, rendering that set of data unusable.

The data presented here generally tend in the expected directions, but some ambiguity is present, especially with respect to usr37 from the Naval Postgraduate School. That user completed one hands-on exercise significantly faster with the baseline capabilities than with the full-KBSA capabilities. Additional analysis of the session log for that exercise had failed to identify any anomalous events (e.g., a long period of inactivity or a system failure of some sort) that might be considered an external corruption of the data.

### 3.4.3. Discussion of the Statistical Tests

Because of the small sample sizes employed in this study, and the unknown nature of their underlying distribution(s), we have used two non-parametric statistical tests to analyze the data: the Sign Test and the Wilcoxon Matched-Pairs Signed-Ranks Test. The latter, because it takes advantage of more information about the relationships within the data, is considered more powerful. A complete description of the underlying rationale and means of performing each test can be found in [Si56]. Appendix C presents abbreviated discussions of the logic, method, and application of each of these tests.

### 3.4.3. Conclusions Regarding the Evaluation of the Hypotheses

With respect to H1 (development productivity), the five data sets analyzed all support rejection of the null hypothesis regarding development productivity at level $p = .05$. We conclude that KBSA technology does have a positive and observable impact on software development productivity.

With respect to H2 (software quality), the combined data sets analyzed support rejection of the null hypothesis regarding software quality at level $p = .05$. We conclude that KBSA technology does have a positive and observable impact on software quality.

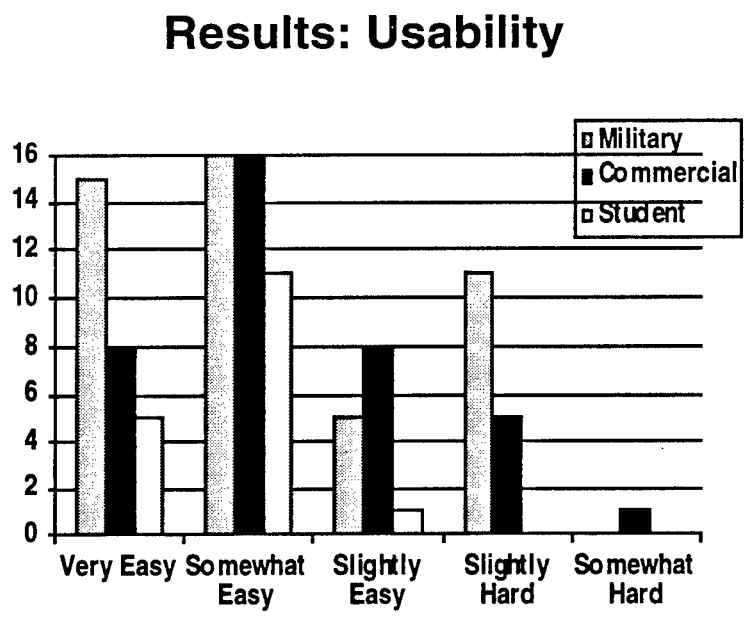
Finally, with respect to H3 (software-time-to-completion), the combined data sets analyzed do not support rejection of the null hypothesis regarding time to completion at level $p = .05$. We conclude that this study has been unable to show that KBSA technology has a statistically significant positive and observable impact on software time-to-completion.

## 4. Discussion of Findings

### 4.1. Usability of the Concept Demonstration System

We collected quantitative data on usability via survey forms that each subject completed after each hands-on exercise. Further, we could not help but form subjective impressions as we observed the subjects using the Concept Demo. In general, our subjective impression was that the subjects were able to use a specific feature of the Concept Demo after a half-day of training[3]. For example, they were able to load a knowledge-base, generate and manipulate several diagrams depicting the hypertext node structure (or a class hierarchy, or an E/R diagram), and navigate through these structures successfully. After two days, they were generally able to navigate between graphical presentations fo formal requirements and hypertext expressions of informal requirements, and to perform typical development and maintenance tasks (via evolution transformations).

While we did not formulate any hypotheses on Concept Demo usability, the data presented in the previous section demonstrate that the Concept Demo's direct manipulation interfaces can, in general, be used with relative ease and material success by subjects with as little training as two days.




[3] This is not true of the pilot subjects at AFIT. Based on the initial difficulty that they encountered, we revised several elements of the training program and added an introductory "Working With the Concept Demo Interface" hands-on exercise. This appeared to make a great deal of difference in the experience of subsequent experimental subjects.

On the other hand, the subjects clearly felt that the Concept Demo could be made far more usable. We encouraged them to give us feedback on how the its usability could be improved, and they were not shy about doing so. Appendix B presents a summary of their comments. Here, we note that the bulk of the subjects' comments can be organized into three major categories: (1) Concept Demo implementation issues, such as response time and performance; (2) KBSA functionality issues, such as suggestions for new features; and (3) issues dealing with the design of the Concept Demo interface itself.

### 4.2 KBSA Impact on Developer Productivity

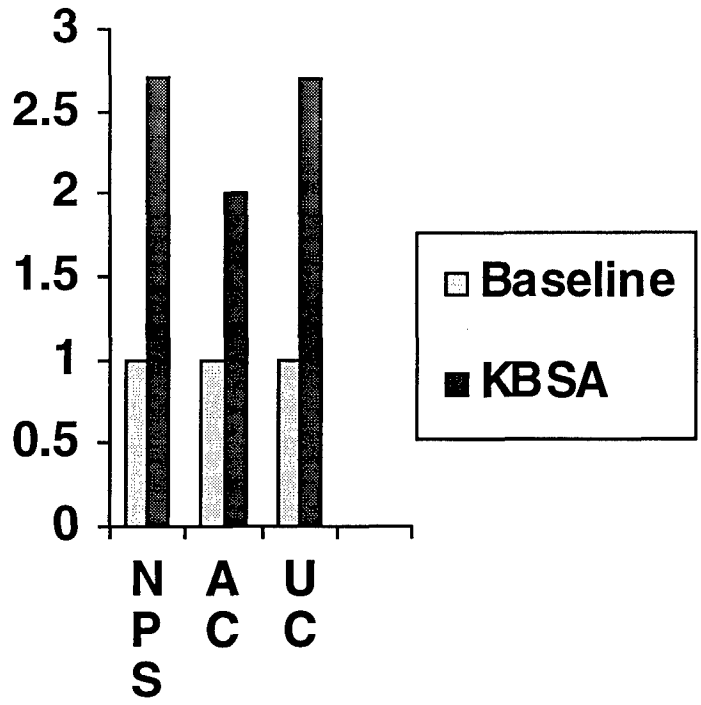
The data presented in section 3 (above) also support the claim that KBSA technology improves developer productivity. Over a series of 99 observations, full KBSA technology supported the completion of the task in less time (than baseline technology) in 96 cases. These differentials were often on the order of 300%. In other words, task performance supported by the baseline OO-CASE technology required four times longer than that required when advanced KBSA technology was available.

Using a nonparametric statistical technique, the Sign Test ([Siegel p. 75] and described in detail in Appendix C), we can reject the null hypothesis at a level of significance of .05, and thus accept the alternate hypothesis that KBSA technology has a positive impact on developer productivity.

This conclusion must be qualified in several important dimensions. First, our data was captured in an experimental setting, not in an actual development context. To some extent, however, this qualification is itself counterbalanced by the reports of field use of KBSA-like technology reported earlier in section 2 (above). Second, our claim of increased productivity pertains to the development and maintenance of formal specifications, rather than to the full life cycle of software development and maintenance. Again, the reader must determine the degree to which he or she is willing to consider that this qualification is countered by the field experience reports.

This positive impact was especially impressive when one considers the higher-order evolution transformations, such as Attribute-to-Relation and Bundle. The typical observed impact of these two transformations was often on the order of 200% (indicating a reduction of 3 to 1) to 400% (indicating a time reduction of 5 to 1).

These "reduction factors" offer an alternative view of these results. Analysis of the normalized data presented in section 3 (above) generates the following reduction factors for specific full-KBSA transformations and for full-KBSA technology overall.




| | NPS | AC2 | UC1 | Average |
|---|---|---|---|---|
| Bundle | -185% | -169% | -314% | -220% |
| Attribute-Rel'n | -192% | -104% | -191% | -168% |
| Create-Inverse | -125% | -3% | -43% | -84% |
| Full-KBSA | -166% | -103% | -168% | -153% |

TABLE 4.2-1
COMPARISON OF AVERAGE IMPACT BY SUBJECT GROUP
AND TRANSFORMATION TYPE

The values presented here are derivative from tables presented in earlier sections. For example, the -185% average impact of the Bundle transformation among the NPS subjects is derived from the data presented in rows labeled "Bundle" in Tables 3.4.2.2-1 and 3.4.2.2-2. Similarly, the AC2 average values are dervied from the specific data in Table 3.4.2.2-3, and the UC averages derived from the data presented in Tables 3.4.2.2-4 and 3.4.2.2-5. Please note that, since the number of observations varied significantly between subject groups (i.e., NPS, AC2, and UC), the values presented here in the column labeled "Average" represent the average across the full set of individual observations, not the average of the three subject-group averages presented in this table itself. Thus, to derive the overall average impact of the Bundle transformation (-220%), one has to refer to the "Bundle" rows in all of the tables presented in section 3.4.2.2 above.

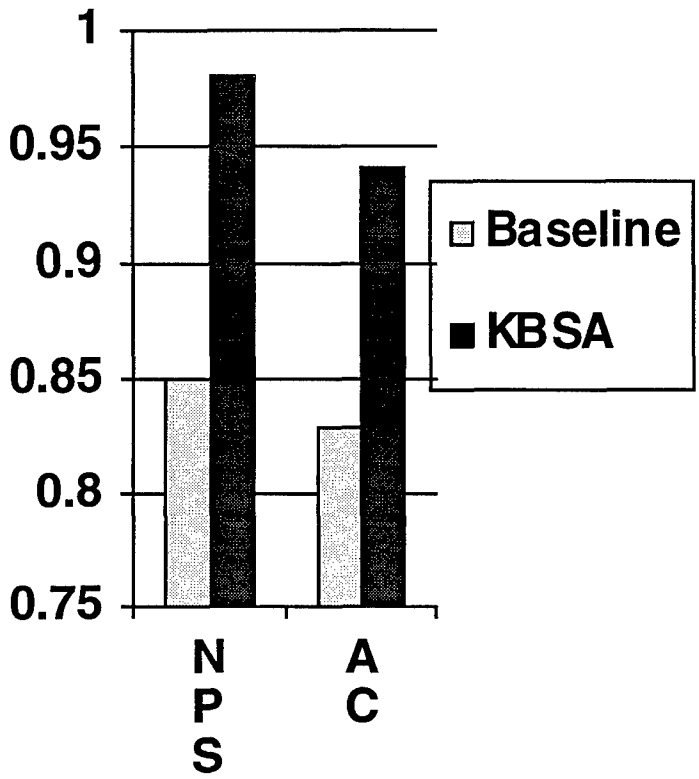
However, for several reasons, both of these views of the data probably represent conservative estimates of the impact of KBSA technology. First, it was fairly difficult to define the baseline capability, and we chose to be liberal in doing so. That is, we consistently included functionality in the baseline when we encountered differing opinions as to whether that functionality was present in state-of-the-market OO-CASE technology today. Thus the baseline that we tested against was a strong baseline. Second, we encountered certain technical constraints on the degree to which we could disable certain KBSA functionality in implementing the baseline version. Thus, short of building a completely new implementation of the Concept Demo, which was clearly outside the scope of this project, we were forced to compare the advanced KBSA technology against a strong implementation of a strong baseline functionality. Finally, our subjective experience leads us to subscribe to the theory that KBSA technology is additive or cumulative beyond the degree to which we could test. For example, our subjects made minimal use of the capabilities of the Concept Demo's Development History mechanism, since we did not attempt to have them "simulate development errors" or to simulate parallel development paths to be merged into a single, consistent specification. We believe that the strong synergy that exists between those full-KBSA technologies we tested and others we did not test will further increase this initially observed impact.

The impact of the higher-order transformations has an important implication for developers of future KBSA technology, such as the KBSA Advanced Development Model (ADM). To provide effective support, they will need to pay close attention to the specific development tasks they intend to support, in order to design the appropriate set of evolution transformations. The set of generic evolution transformations is only a set of potential building blocks — analogous to an alphabet. But these building blocks can be flexibly combined in many ways to form the really meaningful units, corresponding to words, sentences, and paragraphs in language. The important question, however, concerns the nature of those combinations — they must be appropriate to the needs of the KBSA-supported developers. In our experimental setting, we had the luxury of defining tasks that were appropriate to the pre-existing advanced transformations (e.g., Bundle); in field use of the ADM and subsequent KBSA technology, that will not be a viable option. So the needs and methods of the users must be understood, and appropriate higher-order transformations designed based on that understanding.

### 4.3. KBSA Impact on Software Quality

The actual data relating to the software quality impact of KBSA technology — that is, to the internal consistency and completeness of the underlying formal specification — are presented in section 3.4.2.3 (above).

Using a nonparametric statistical technique, the Sign Test [Si56, p. 75], we can reject the null hypothesis at a level of significance of .05, and accept the alternate hypothesis that KBSA technology has a positive impact on software quality. An alternative view of these results is presented in the following diagram.




One potential qualification of this conclusion concerns its external validity: that is, to what extent is the result observed in the experiment likely to be manifest in real-worls software development? In other words, to what degree does the internal consistency of a formal specification can act as an indicator of the quality of a deliverable software application? We strongly believe that (1) a more consistent formal specification enables more efficient and accurate code generation and (2) a more consistent formal specification is a better starting point for code generation. However, we recognize that the generation of executable code from formal specifications is another major step in the KBSA-supported software development process, and that the KBSA impact on software quality of the final deliverable is suggested rather than directly indicated by our experimental data.

Once again, the impact of KBSA technology may here be understated, due to the consistency checking functionality enabled by the underlying formal

specifications. For example, this functionality would include the Concept Demo's Static Analysis and Resource Analysis capabilities. These were not used in the experimental study, as their use involved either the (demo-level) Agenda Mechanism or writing of formal specifications in ERSLa.

### 4.4. KBSA Impact on Time to Completion

The actual data relating to the development period impact of KBSA technology — that is, to the time to completion for development of a formal specification — are presented in section 3.4.2.3 (above).

As discussed in Appendix C, neither of two nonparametric statistical techniques, the Sign Test [Si56 p. 75] and the Wilcoxon Matched-Pairs Signed-Ranks Test [Si56, p. 83] enables us to reject the null hypothesis at a level of significance of .05. Therefore, we conclude that this study has been unable to demonstrate that KBSA technology has a statistically significant positive and observable impact on software time-to-completion.

## 5. Conclusions and Lessons Learned

### 5.1 Conclusions

5.1.3. KBSE technology is providing value in industry use today. Companies like Andersen Consulting , Boeing Computer Services, IBM/Canada, Motorola, and AT&T Bell Labs are taking advantage of the potential of this technology, and are presenting their results in public discussion.

5.1.2. KBSA technology demonstrates an experimentally observable positive impact on software development productivity. This is consistent with the observations of field use, especially the experiences reported by Andersen Consulting.

5.1.3. KBSA technology demonstrates an experimentally observable positive impact on software quality. This is consistent with the observations of field use, especially the experiences reported at Motorola.

5.1.4. KBSA technology is usable in its current state. Our subjects, including software development professionals, military personnel, and graduate students, learned to use a powerful subset of the capabilities of the KBSA Concept Demonstration System effectively within a two-day training period.

### 5.2 Lessons Learned

5.2.1. Software experimentation is hard, but it can be done. As other studies have suggested, our experience emphasizes the need to control for different individual skills and learning rates in the experimental design.

5.2.2. Higher-order evolution transformations, such as the Bundle and Attribute-to-Relation transformations, provide biggest productivity impact, but are perhaps most domain-specific. This implies the need to study the specific needs of users as a basis for designing these transformations, at least until we have established a set of generally applicable higher-order transformations.

5.2.3. This study has highlighted the importance of usability aspects in the design and implementation of KBSA technology. The suggestions made by the experimental subjects regarding usability indicate that KBSA technology has further to go in this dimension: KBSA is usable today, but it can be far more usable in the future.

## 6. Bibliography


[AC92} Andersen Consulting. Software Test Description: Knowledge-Based Software Assistant concept Demonstration System. CDRL A007, Government Contract F30602-89-C-0160. September, 1992.

[Ar94] Arnold, J. "Toward Collaborative Software Processes." (draft) position paper for the 9th International Software Process Workshop (1994).

[BH91] Buss, E., and J. Henshaw. "A Software Reverse Engineering Experience." IBM/Toronto technical report TR-74.065: September, 1991.

[BH92] Buss, E., and J. Henshaw. "Experiences in Program Understanding." IBM/Toronto technical report TR-74.105: July, 1992.

[BSL93] Burton, S., K. Swanson, and L. Leonard. "Quality and Knowledge in Software Engineering." AI Magazine: Winter 1993.

[De92a] M. DeBellis, K. Miriyala, and W. C. Sasso. "Demonstration Description: The KBSA Concept Demonstration System" in W. Lewis Johnson, ed., *KBSE '92: Proceedings of the Seventh Knowledge-Based Software Engineering Conference.* McLean, VA: September 20-23, 1992.

[De92b] M. DeBellis, K. Miriyala, S. Bhat, W. C. Sasso, and O. Rambow. Final Report: Knowledge-Based Software Assistant concept Demonstration System. CDRL A007, Government Contract F30602-89-C-0160. October, 1992.

[DoD91] Department of Defense. Software Technology Strategy (draft). DDR&E: December, 1991.

[Gr83] C. Green, D. Luckham, R. Balzer, T. Cheatham, and C. Rich. Report on a Knowledge-Based Software Assistant. RADC-TR-83-195 (August 1983).

[HTB93] Henshaw, J., J. Troster, and E. Buss. "Filtering for Quality." IBM/Toronto technical report TR-74.132: September, 1993.

[Ko92] Kotik, G., and L. Markosian. "Knowledge-Based Software Reengineering Tools" in W. Lewis Johnson, ed., *KBSE '92: Proceedings of the Seventh Knowledge-Based Software Engineering Conference.* McLean, VA: September 20-23, 1992.

[Le89] Lee, A. "A Scientific Methodology for MIS Case Studies." MIS Quarterly, March 1989.

[Le92] Levenson, N. G. "High-Pressure Steam Engines and Computer Software" in L. A. Clarke and C. Ghezzi, eds., *Proceedings of the 14th International Conference on Software Engineering:* Melbourne, Australia (May 11-15, 1992).

[Mo92a] Moseman, L. K., II. "View From the Secretariat: Metrics Are Meaningful" in *Crosstalk* issue number 35 (August 1992), p. 12.

[Mo92b] Moseman, L. K., II. "Improving Software Quality Through Measurement" in *Crosstalk* issue number 36 (September 1992), pp. 2-5.

[RL92] Rome Laboratory. KBSA Advanced Development Model (ADM). RFP F30602-92-R-0039 (June, 1992).

[Sa90a] Sasso, W. C. "Empirical Study of Re-Engineering Behavior: Design Recovery by Experienced Professionals." Software Engineering: Tools, Techniques, Practices 1(1): March 1990.

[Sa90b] Sasso, W. C. Four Dimensions of Project Management: Design, Engagement, Administration, and Facilities. Andersen Consulting Internal Report: October, 1990.

[Sa91] Sasso, W. C. Refine Users Study: Preliminary Summary of Findings. Andersen Consulting Internal Report: May, 1991.

[Si56] Siegel, S. Nonparametric Statistics for the Behavioral Sciences. McGraw-Hill. New York: 1956.

[TSL] Terveen, L., P. Selfridge, and M. D. Long. "'Living Design Memory' — Framework, Systems, Lessons Learned." draft paper, currently under review for publication.

[Tr92] Troster, J. "Assessing Design-Quality Metrics on Legacy Software." IBM/Toronto technical report TR-74.103: September, 1992.

[Vo93] Votta, L. "Comparing One Formal to One Informal Process Description." position paper for the 8th International Software Process Workshop (1993).

[We94] Weigert, T. Personal Communication. December, 1994.

**KBSA Concept Demonatration**
**Trainingand Data Collection Program Agenda**

1. General introduction to KBSA (classroom)
    - Focus of the evaluation
        - Usability of transformations
        - Experiences with transformation-based development
    - Current status of KBSA
        - Brief history of KBSA
        - Concept Demo
        - Advanced Development Model
        - AC: relationship to other AC development tools
    - Explanation of the training and exercises
        - Artificial software development process
        - CD not completely functional, so scope of exercises is limited
        - Cooperative attitude requested (don't overwrite objects)
        - Advantages of CD access
    - Completion of trainee experience survey

2. Lecture/discussion: CD User Interface (classroom)
    - CD user interface conventions
    - Baseline process model for class hierarchy definition
    - CD session manager/log

3. Hands-on session 1: warm-up
    - Using the CD interface
    - Browsing existing CD artifacts (e.g., hypertext)
    - Defining a class hierarchy (baseline version)
    - Starting/stopping the session log

4. Lecture/discussion: CD Process Model (classroom)
    - High-level model overview
    - Intermediate process model description: Domain Specification
    - Baseline process model for E/R manipulation
    - Baseline process model for traceability

5. Hands-on session 2: Basic specification development
    - Adding relations and attributes in an ER diagram
        - baseline attribute/relations transformations
    - Creating traceability links between objects and hypertext
        - baseline traceability transformations

6. Hands-on session 3: base uability data collection
   - Defining a class hierarchy (as above in 3)
   - Adding relations and attributes in an ER diagram (as above in 5)
   - Creating traceability links between objects and hypertext (as in 5)
   - CD analysis features automate evaluation of the threshold criterion

7. Lecture/discussion 3: Building a domain specification
   - Using transformations to build a domain specification

8. Hands-on 4: Building a domain specification
   - Hands-on practice with building a domain specification

9. Hands-on 5: Usability data collection
   - Session log initiation
   - Building a domain specification
   - Save kb-module
   - Analysis of product and session log

10. Lecture/discussion 4: Maintaining an existing domain specification
    - Application of KBSA functionality for domain spec maintenance
    - Walkthrough of maintenance task

11. Hands-on 6: Maintaining an existing domain specification
    - Hands-on practice with maintenance task

12. Hands-on 7: Usability data collection
    - Session log initiation
    - Maintaining a domain specification
    - Save kb-module
    - Analysis of product and session log

13. Lecture/discussion 5: Adapting an existing domain specification
    - Application of KBSA functionality for adaptation
    - Walkthrough of adaptation task

14. Hands-on 8: Adapting an existing domain specification
    - Hands-on practice with adaptation task

15. Hands-on 9: Usaability data collection
    - Session log initiation
    - Adapting a domain specification
    - Save kb-module
    - Analysis of product and session log

13. Lecture/discussion 6: De-briefing
    Questions and answers
    Subjective feedback from trainees

14. Hands-on 10: Unstructured experimentation
    An opportunity for hands-on CD experimentation (optional)

## Subjects' Usability Feedback

This report presents feedback from Concept Demo users who have participated in the EE PRDA training and evaluations. This feedback has been screened, and comments specifically related to the Concept Demo implementation have been removed. 103 comments remained; they can be subjectively organized into six major categories:

1. Better presentation of system status information (11 comments): AFIT-1, AFIT-5, AFIT-8, AC1-6, NPS-15, AC2-1, AC2-13, AC2-32, UC-2, UC-17, and UC-18.

2. Help finding, manipulating, and interpreting specific objects, mainly display-level objects such as windows and icons (18 comments): AFIT-3, AFIT-4, AFIT-9, AC1-1, AC1-10, AC1-13, NPS-4, NPS-6, NPS-9, AC2-7, AC2-9, AC2-27, AC2-28, UC-5, UC-7, UC-8, and UC-19.

3. Discrepancies from accepted or familiar interface standards (16 comments): AFIT-2, AC1-2, AC1-3, AC1-8, AC1-9, NPS-1, NPS-5, NPS-10, NPS-13, AC2-2, AC2-3, AC2-8, AC2-12, AC2-20, AC2-22, and AC2-26.

4. More intelligent assistance for users (12 comments): AFIT-6, AC1-7, AC1-12, NPS-14, NPS-23, AC2-6, AC2-10, AC2-11, AC2-17, UC-1, UC-6, and UC-20.

5. Difficulty of use or ease of use (27 comments): AC1-4, AC1-5, NPS-3, NPS-7, NPS-8, NPS-11, NPS-16 through NPS-22, AC2-4, AC2-14, AC2-18, AC2-23, Ac2-25, AC2-31, UC-10 through UC-16, and UC-21.

6. Other comments (19 comments), including response time issues (2 comments: AFIT-7, and UC-3); extensions to the existing Concept Demo functionality (3 comments: AFIT-11, AC2-21, and UC-4); understanding the Concept Demo's conceptual design or process model (4 comments: AC2-5, AC2-16, AC2-24, and UC-9); compliments (6 comments: NPS-12, NPS-24 through NPS-26, AC2-15, AC2-19; and preferences for more keyboard-driven (rather than mouse-driven) interaction (4 comments: AFIT-10, NPS-2, AC2-29, and AC2-30).

This categorization, while not perfect, suggests that users of KBSA technology pay attention to (at least) the following five principles of human/computer interaction:

I. Help me (the user) understand the current status of the KBSA system. I need to know whether I should wait for the operation to complete, or whether I should take some action because the system is in a state of malfunction. I especially need to know when the system is waiting for me (e.g., to enter some data).

II. Design the interface itself to help me use it; use different icons, shapes, colors, and naming conventions to help identify different types of objects (e.g., a formal spec versus a hypertext node, or a user-creed link versus a system-created link). Give me a mechanism that provides direct-access to an object if I know its name, such as a "Find" facility in a word processor.

III. Use consistent, standard, and familiar diagramming and naming conventions; don't mix the entity/relation model with the object-oriented model, and use a familiar representation of the object-oriented model, not one that is unconventional and unique to the Concept Demo.

IV. The KBSA interface ought to be at least as intelligent as current commercial interfaces: for example, it ought to disable and "gray out" inappropriate evolution transformation selections (such as "Attribute to Relation" if the selected object doesn't currently have any user-defined attributes).

V. The KBSA interface ought to be at least as easy to use as current commercial interfaces: for example, it ought to provide "cut and paste" and "drag and drop" capabilities in support of the specification editing process.

The specific comments below are offered in the hope that they will inform the design and implementation of the ADM user interface, especially those interface elements related to specification development using evolution transformations.

**Comments from members of the AFIT session (7/19-20)**

AFIT-1. Status window messages could be finer-grained: e.g., generating menu, awaiting user's menu selection, and menu aborted would all be more informative than the CD's standard task executing (even when awaiting user menu selection).

AFIT-2. A "quit-CD" option (less brutal than killing the Refine process) would be desirable.

AFIT-3. Provide a "direct access to object" capability; let me enter the object's name and have it displayed in a default manner, or provide a menu of display options.

AFIT-4. Allow a customizable user interface, so that each user can define window display priorities (e.g., "I want the E/R window on top at the end of every operation").

AFIT-5. We appreciated the option of turning off the display of all kb-changes in the output window.

AFIT-6. Consider a MacIntosh-like menu capability that disables and grays out operations when they cannot be performed. For example, disable/gray out "Attribute to Relation" for entities without any attributes currently defined.

AFIT-7. If response time is slow[1], then consider adding a buffer into which a stream of operations could be entered, and then executed at the system's leisure.

AFIT-8. Where it is possible to generate more specific error messages (i.e., more specific than "Task not completed"), it would be appreciated.

AFIT-9. A "Find window" or "List currently displayed windows" option would be appreciated.[2] One AFIT participant suggested that a multi-screen display option be considered.

AFIT-10. Consider the use of key-commands (e.g., CTL-X to cut/delete), as an alternative to cascading menu selections (primarily for power users).

AFIT-11. Consider implementing support for Extended E/R diagram capabilities.

**Comments from members of the Andersen Consulting session (8/29-30)**

AC1-1. The indication of the active window (green border) is pretty subtle; can it be made more easily visible?

AC1-2. I found the naming conventions somewhat unconventional (e.g., "transformation" and "abort").

AC1-3. Dialog window conventions are non-standard. For example, the dialog window is not active unless the cursor is within the window frame. This is different from the click and sustain convention used in PC GUI-based OSs.

AC1-4. I would like to be able to mark a hypertext string and then say "define a class with this string as its name." This would eliminate some typing errors, and save a few interaction steps.

AC1-5. I couldn't find a "cut and paste" option that would enable me to move a group of subclasses within the class hierarchy. I looked for one when I defined several subclasses of class X, and then realized that I had intended to define them as subclasses of class Y.

---

[1]AFIT was running the CD on Sparc2s. However, the Andersen subjects running on Sparc10s also experienced slow response time, and often "got ahead of the CD interface."

[2]The AFIT users tended to want to have six or more large windows open at a time, so they could see all the different things that were happening. This led to stacks of windows, that they had to sort through after each transformation.

AC1-6. I'd like to have more detail/better explanations in the error message presented in the Status window.

AC1-7. I think there should be a more dramatic sign to show the completion of each operation so that users can't initiate another operation prematurely, causing system errors.

AC1-8. The distinction between user-created relationships and system-maintained links (e.g., "created-by" or "formalized-by") should be evident in the diagrams. For example, use a different types of arrow or a dashed line for system maintained links.

AC1-9. Would suggest adoption of window and mouse movement conventions based on Windows (or some other appropriate) standards.

AC1-10. Objects, classes, hypertext objects, etc. should be displayed using different icons to facilitate recognition.

AC1-11. When reshaping a window, the icons in it should be re-displayed proportionately without forcing the user to zoom in or out. Or else a combined reshape/zoom command, since we always do them in sequence.

AC-12. When defining a new attribute, the range type should have a default value, or else the operation should check to ensure that the definition has been completely specified before "Do It" is enabled. Tab keys should be available to move from field to field in data entry boxes with multiple fields.

AC1-13. I'd like to have a "find object" command to navigate large Semantic Net Diagrams with lots of nodes and links displayed.

**Comments from members of the Naval Postgraduate School session (10/20-21)**

NPS-1. Commands were too "bulky" — I didn't like the nested command sequences required to do things like zooming.

NPS-2. Commands should be placed where a knowledgeable person could use the keyboard (??).

NPS-3. I experienced some confusion about when to use mouse-left, mouse-middle, or mouse-right.

NPS-4. It would be easier if the program didn't lay new windows directly on top of old windows.

NPS-5. Implement "cut and paste" for adding attributes and relations. It would save the developer a great amount of time and reduce errors.

NPS-6. I confused the Hypertext Graph window and the Used-by Relations window when searching for the Graph Classes function.

NPS-7. When entering multiple subclasses (separated by a space) an error in one caused an error message ("name already exists") but the only apparent method to recover was to abort and re-key the entire list, which was inconvenient.

NPS-8. Default options need to be more easily identified, understood, and accessed.

NPS-9. Diagram layout includes crossing lines, which are difficult to read as complexity increases.

NPS-10. Standard typing conventions (e.g., backspace) should function in the standard way.

NPS-11. Overall an outstanding concept. DoD needs this kind of tool. It needs to be "friendlier."

NPS-12. The second exercise went quickly and was fairly easy. As a new user, I felt fairly comfortable after the first [exercise].

NPS-13. The mouse buttons get confusing: left, middle, right? Windows appear in a place decided by the system, not the last place I placed it. This is frustrating. Class hierarchy screen doesn't seem to fill up. Zooming is a bit unfriendly.

NPS-14. I couldn't remember how to pop-up the ER diagram, and the CD doesn't help me [remember].

NPS-15. The status window could be improved. It is not clear when the program is executing a task when the message "task canceled" is in the window.

NPS-16. "Cut and paste" or "click and drag" capability from the hypertext to the ER model would prevent a lot of re-keying. Is there a way to automate that and have the ER diagrams updated directly from hypertext (i.e., in adding attribute and relation definitions)?

NPS-17. "Click and drag" (or even "highlight and copy") capability would be nice. Relatively easy to use.

NPS-18. A list of options for text entry would be nice ...

NPS-19. Reopening the add attribute window for each attribute is inefficient.

NPS-20. Retyping the attribute name in the add attribute window (vice being able to copy it into the window) is inefficient.

NPS-21. Too much typing! Perhaps a list of options from the hypertext document can be used to access a mouse-selectable set of operations on superclasses or attributes on entities.

NPS-22. It would be great if you could "click and drag" attributes, etc.

NPS-23. It would help significantly if there was a "help" facility.

NPS-24. Really interesting software. It gets more understandable with practice.

NPS-25. Great program!

NPS-26. Overall this is an amazing product. With a better interface, including function keys and a top menu bar, it could be easy to use. I look forward to the ADM.

**Comments from members of the Andersen Consulting session (10/24-25)**

AC2-1. I got ahead of myself [and the Concept Demo] in entering commands. I found myself rarely looking at the Status Window. Maybe it would be useful to highlight the Status Window when a task is being executed. Should/could there be a mechanism to show how many tasks are pending (if more than one)?

AC2-2. Don't use "abort" as the term for closing a menu; use "close" or "exit" or "cancel." Abort sounds like you are killing the application (i.e., Space Shuttle Abort).

AC2-3. On menus, give some indication for those selections that will bring up another menu; this tells the user what to expect navigation-wise if he makes the selection. For example, use the menu selection <Open ...> rather than <Open> if the selection of open will generate another menu.

AC2-4. I am still unclear as to the functions of the three mouse keys. It is not intuitive what functions lie under each mouse-key (l,m,r).

AC2-5. I had difficulty in understanding the differences between hypertext nodes and their definitions and class definitions [for classes described in the hypertext node]. Good discussion made these differences clearer.

AC2-6. In renaming an object (e.g., a subclass), it may be helpful to default to the current spelling rather than typing the entire word again. Most spelling errors are quite simple, a character or two. This could speed things up a bit.

AC2-7. As a user-selectable interface feature, when new subclasses are added, the Graph Classes window could automatically zoom out to that all sub-classes can be seen on the screen.

AC2-8. Menus are extremely cluttered and could be better organized and partitioned.

AC2-9. It seemed unclear whether the links were actually made to the attributes as they were not visible as they could not be viewed in the Semantic Net Model. Otherwise the exercise was good, ...

AC2-10. ...except for having to constantly delete what is in [attribute-name] fields [in the Add Attribute dialogue box].

AC2-11. Relation-range field should initialize to blanks when defining a relation.

AC2-12. Dialogue boxes in the Semantic Net Model are too small to hold text (or the text is too large).

AC2-13. CD System needs [to display] "confirmation on deletion.

AC2-14. Tool needs to be simplified in its interface. There are too many different commands and it's not intuitive how to access them.

AC2-15. Showed how the extended features [full KBSA functionality] made the work easier. Again the interface is intuitive, and the tasks, though repetitive, remained interesting.

AC2-16. Again, the task was easily understandable, with some small problems on "bundling."

AC2-17. The attribute to relation and bundle operations are quite easy to use. The only point that may be important is that I tended to forget to put the target entity [of attribute to relation] into the ER diagram, which wasted time.

AC2-18. Hint: for the most likely users of this tool, some of the nomenclature can be more intuitive (rather than computer science jargon).

AC2--19. Overall the CD is a powerful tool, but whether or not its unique functions can be applied to real-world issues remains the test.

AC2-20. I want to do "drop and drag" for hooking things together and sorting out classes. Not menus. Prefer this as [it is] used in a lot of CASE tools rather than menus.

AC2-21. [It would be] nice to have a tool box or GUI front end from which you could select icons for classes, types of lines to show relations, etc.

AC2-22. Be sure to use standard OO terminology and conventions: otherwise product will not be acceptable in the Object World.

AC2-23. It was not immediately apparent how we could get rid of an attribute. After we saw how it works, it's okay but the way it is done is still perceived as cumbersome [?] and not intuitive.

AC2-24. Given the wide array of [possible ways of ] doing certain operations, is there a recommended method? Could you integrate [?] the various methods of accomplishing the same operation?

AC2-25. Overall, I get the feeling that the CD model is more difficult than it could be with regard to maintenance. Maintenance should be equivalent to or easier than creating new objects in ease of use since it is probably done more than creation.

AC2-26. Do not care for having to press return prior to using mouse to move to the next data entry field.

AC2-27. Still have difficulty remembering from which model one can perform which functions. Would be helpful if this were more consistent ... do not feel that, due to this, this [tool] is terribly easy to master.

AC2-28. It would be nice to have different types of objects (e.g., hypertext versus class definitions versus relation definitions) indicated in displays with different types of display icons.

AC2-29. The interface is fairly intuitive, but it would be nice to not be so "mouse-dependent" but be able to use the keyboard more.

AC2-30. Use hot keys, field tabs ...

AC2-31. The back-end implementation of the KB etc. should be hidden from the designer/programmer. OO has a huge learning curve and the terminology [e.g., menu selections, window titles] used should reflect the vocabulary of the Object World, not the KB world. KB objects do not always map to system/C++ objects ... why make me aware of KB objects that don't map to my business models/object models? Use OO terminology/vocabulary in the presentation layers. Use the OT paradigm, not the KB one.

AC2-32. When adding a new attribute in the Semantic Net Model, if the attribute-domain is incorrect the status window displays "error occurred during task ...

continue." To the user, the root of the error is not mentioned, and thus it may be unclear exactly what is wrong.

**Comments from the University of Cincinnati session (12/21-22)**

UC-1. Should have warned us that an object with that name already exists when we rename another object with a name that is already taken.

UC-2. Pointer icon should change to a stopwatch when CD is executing.

UC-3. There must be a way to improve response time (like keeping the E/R diagram uncluttered).[3]

UC-4. Add functionality to support user-defined evolution transformations.

UC-5. Too many pop-up windows to follow.

UC-6. Like it better when a name is not supplied in the define attribute window.

UC-7. The diagrams should auto-zoom.

UC-8. Mouse-left always brings a window forward when one might want to keep it in the background :-).

UC-9. The Requirements Traceability Diagram doesn't automatically update (like the Semantic Net Model diagram does) making the former less useful.

UC-10. Provide the ability to remove all of a particular kind of link from the current display window (and to keep it removed).

UC-11. Provide the ability to define multiple attributes at once (analogous to the "Add Subclass" function in the Class Hierarchy diagram).

UC-12. Provide the ability to clear links and keep them cleared after new attributes are added.

UC-13. Eliminate necessity of hitting return-key in data entry boxes.

UC-14. Currently active window is always pulled forward, even when you don't want it to be (forward).

---

[3] The University of Cincinnati subjects were running the Concept Demo on Sparc Classics with small amounts of memory and sub-minimal swap space allocations. Performance was painfully slow there.

UC-15. Ability to move a group of subclasses to a new superclass would be helpful.

UC-16. Cut and paste functionality (e.g., from hypertext to graphs) would be helpful.

UC-17. You'd better give more (descriptive) error messages because users may not understand the underlying data model precisely.

UC-18. Need more informative error reporting.

UC-19. The way that diagrams automatically reset their position (after it has been manually set only a few moments before) is annoying.

UC-20. When defining a new relation that has the same name as an existing relation, the Concept Demo should give a warning.

UC-21. When selecting a range parameter with the mouse, the desired parameter may not be visible. If that is the case, allow the user to click on "nothing" and interpret that as a request for the Concept Demo to bring up a data entry box.

## DESCRIPTION OF THE STATISTICAL TESTS

### C-1. The Sign Test

The sign test assumes that we have a set of related samples, and that we can determine the direction of the difference between treatments for each sample. Our data fulfills that criterion in that we have the timestamped session-log measure of how long it took to perform a given task working with the full set of KBSA technology, versus how long it took to perform that task with the baseline technology.

If there is really no difference between the technologies, as stated in the null hypothesis, we would expect these differences to be distributed in random fashion. We would anticipate that the baseline times would be greater than the full-KBSA times in about half the cases, and vice versa in the other half. In probabilistic terms,

$$P(\text{Baseline} > \text{Full-KBSA}) = P(\text{Full-KBSA} > \text{Baseline}) = .5$$

In other words, the probability that the baseline time will be greater than the full-KBSA time is equal to the probability of the inverse; both have probability equal to .5. According to the null hypothesis, this should be true. But if we observe a consistent pattern of time differences (e.g., that the baseline times are consistently larger than the full-KBSA times), we will have to reject the null hypothesis. As described in Siegel ([Si56], p. 75), these are the steps in the application of the Sign Test:

A. Determine the sign of the difference between the members of each pair.

B. By counting the number of pairs, determine the value of N (equal to the number of pairs where the difference is non-zero).

C. The method for determining the probability associated under of a value as extreme as the observed value of x depends on the size of N:

a. If N is 25 or smaller, a table is used to determine the probability of observing x in a sample of size N under $H_0$.

b. If N is larger than 25, z is computed according to the following formula:

$$z = ((x \pm .5) - .5 * N)/(.5 *(N^{**}.5))$$

Reference to another table enables the determination of the probability of observing that value of z under $H_0$.

| H1 Data Pairs | Value of N | Threshold (x) | Value of x | Conclusion |
|---|---|---|---|---|
| NPS T7/T5 | 30 | n/a | 30 | Reject $H_0$ |
| NPS T6/P7 | 20 | 15 | 20 | Reject $H_0$ |
| AC2 T6/P7 | 22 | 16 | 20 | Reject $H_0$ |
| UC T5/T7 | 17 | 13 | 16 | Reject $H_0$ |
| UC T6/P7 | 10 | 9 | 10 | Reject $H_0$ |

| H2 Data Sets Combined | Value of N | Threshold (x) | Value of x | |
|---|---|---|---|---|
| NPS/ AC2 | 8 | 8 | 8 | Reject $H_0$ |

| H3 Data Sets Combined | Value of N | Threshold (x) | Value of x | |
|---|---|---|---|---|
| NPS/AC2 | 8 | 8 | 6 | Cannot Reject $H_0$ |

C-2. The Wilcoxon Matched-Pairs Signed-Ranks Test

The Sign Test considers the direction of the difference between pairs of related data. If the size or magnitude of the difference can be considered, a more powerful test can be employed. The Wilcoxon Matched-Pairs Signed-Ranks Test weights the relative importance of test cases, to give more weight to those pairs exhibiting a large difference between the two conditions. In the context of this study, this test will emphasize cases where a large difference in the task performance time is observed, regardless of the direction of that difference. Because of the small number of observations available for the H3 data set, we will apply this test to them. As described by Siegel (1956, p. 83), the test is performed in this manner:

A. For each matched pair, determine the signed difference $d_i$ between the two scores.

B. Rank these $d_i$ without respect to sign. With tied values of $d_i$, assign the average of the tied ranks.

C. Assign to each rank the sign (+ or -) which it represents.

D. Determine T = the smaller of the sums of the like-signed ranks.

E. By counting, determine N = the total number of $d_i$ having a sign.

F. The procedure for determining the significance of the observed value of T depends on the size of N: either reference to a table or computation and reference to a table is required.

| H3 Data Sets | Value of N | Threshold of Rank Total | Rank Total | |
|---|---|---|---|---|
| Combined NPS/AC2 | 8 | 4 | 7 | Cannot Reject $H_0$ |

*U.S. GOVERNMENT PRINTING OFFICE: 1997-509-127-61021

# *MISSION OF ROME LABORATORY*

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Material Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.